

1/33

10
↓

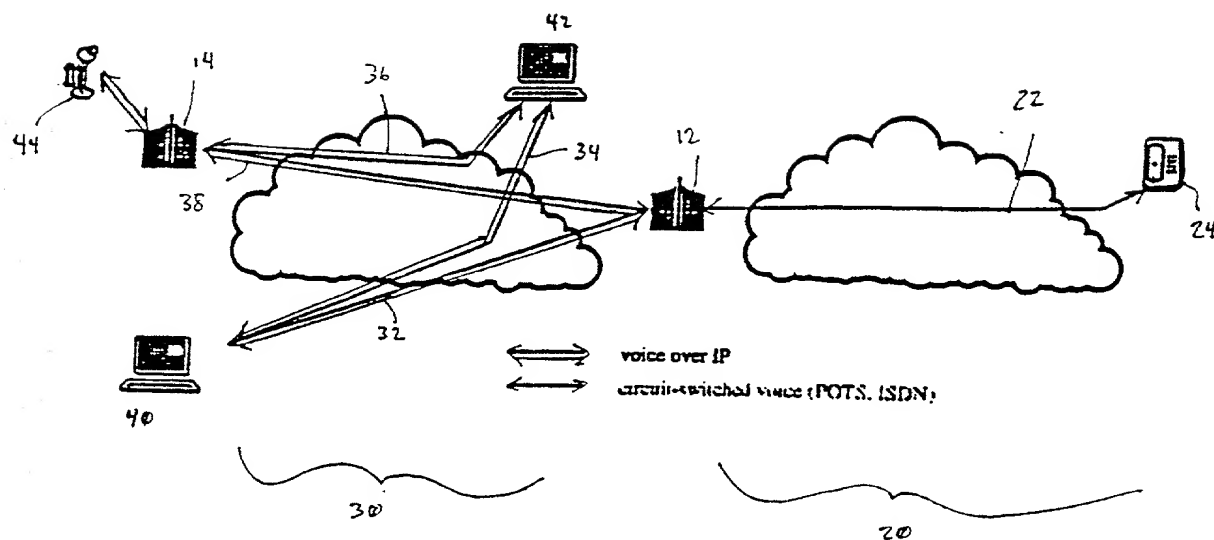
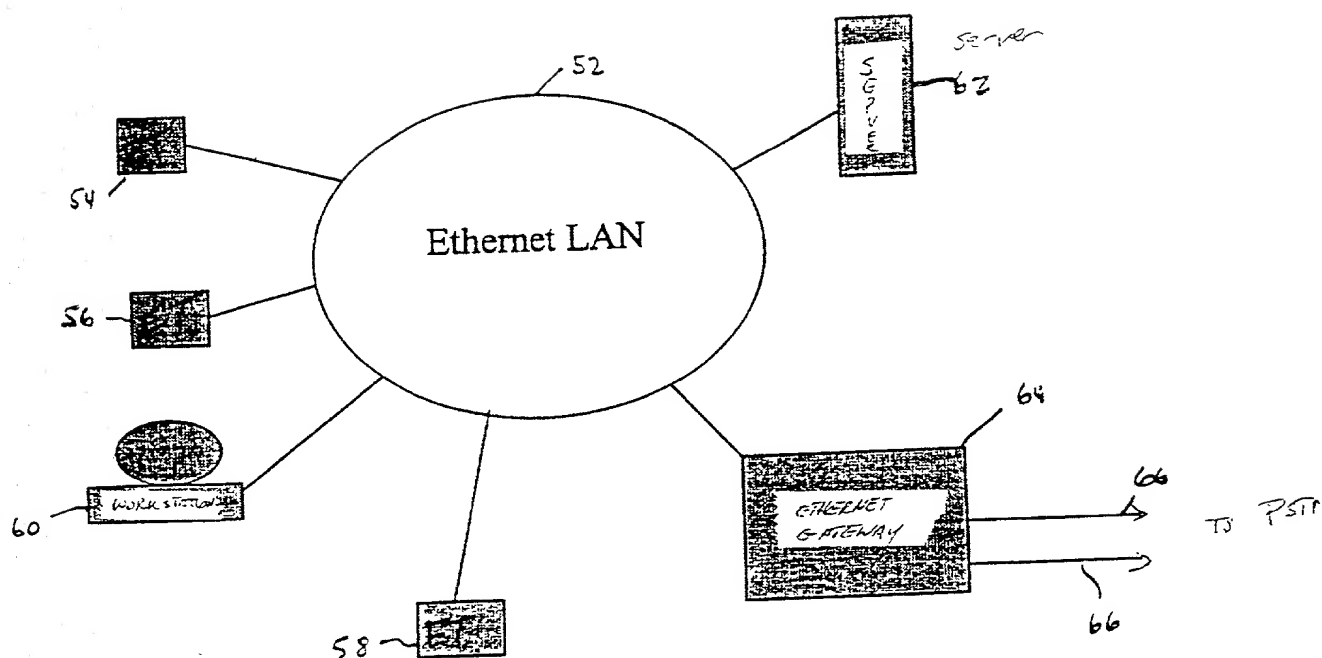


FIG. 1

09/980885 032200

2/33

50FIG 2

3/33

2022033003650

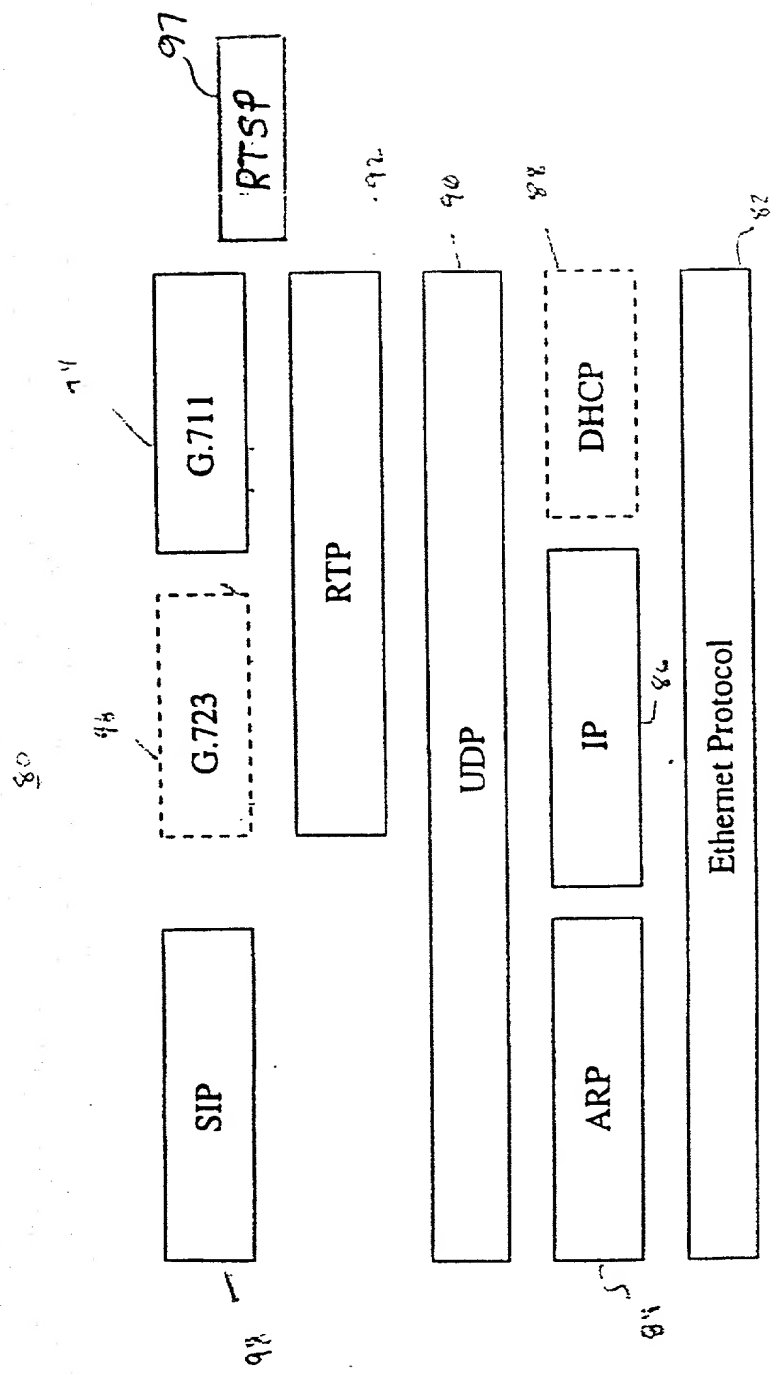
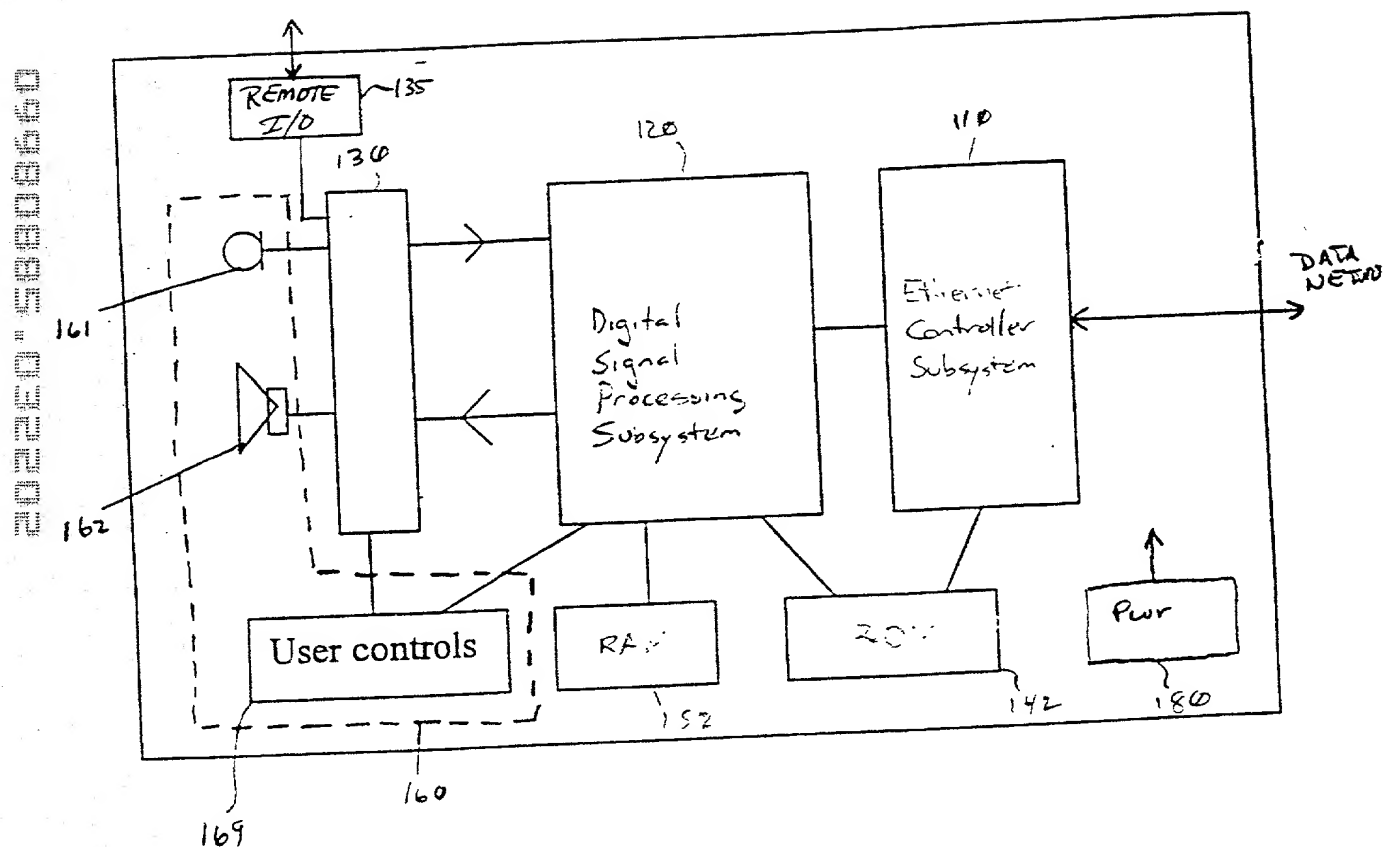
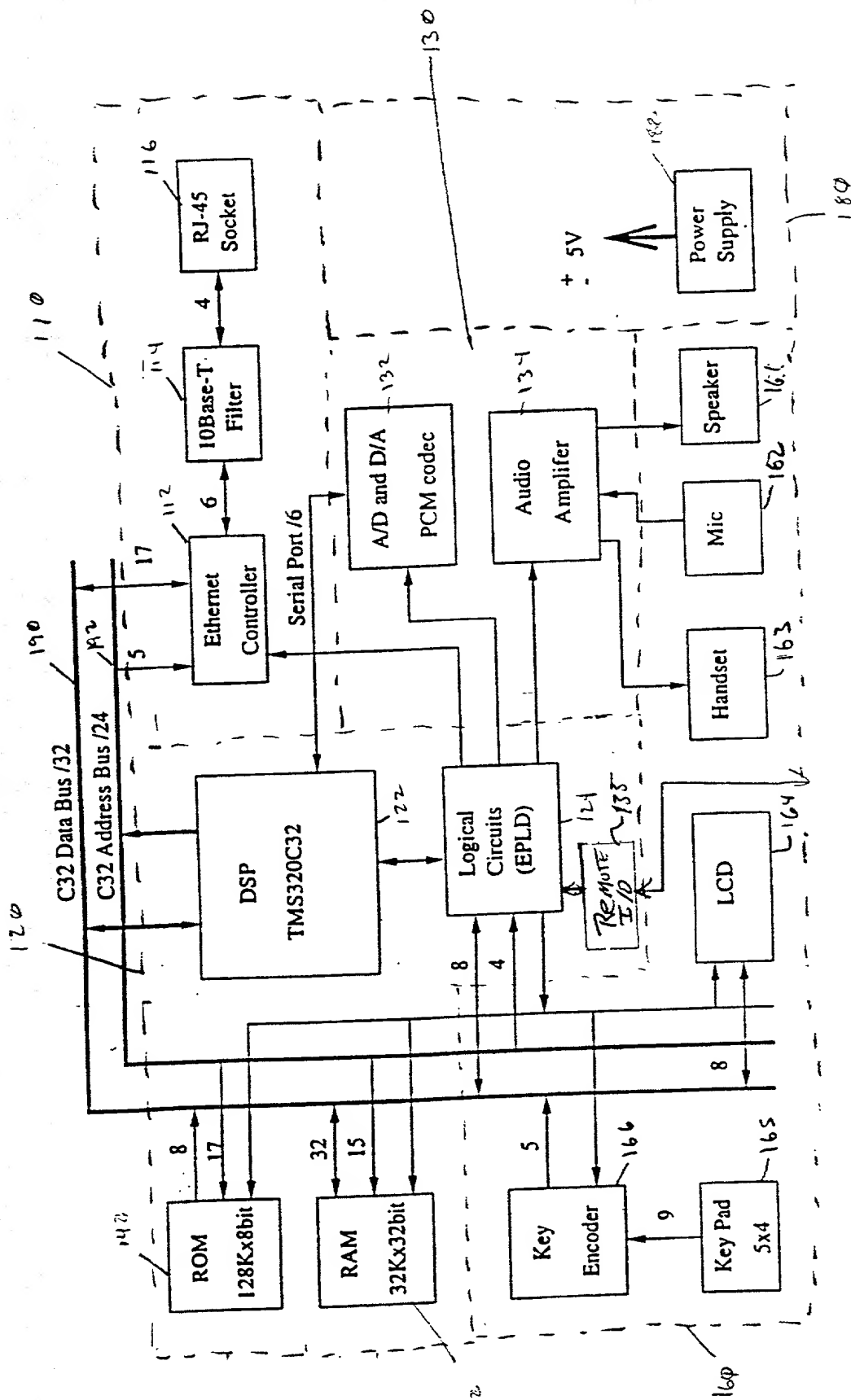


FIG. 3

4/33

100FIG. 4



4/33

DSP Address	Length (hex)	(dec.) Usage
0x1000	0xB000	128K ROM
0x20000	0x10	16 LCD
0x810000	0x20	32 Ethernet controller
0x820000	0xf	16 Keypad read, audio amplifier, control, hook control, system software reset
0x87fe00	0x200	512 Internal RAM
0x900000	0x8000	32K External Ram

Fig. 6 DSP memory map

09980885-03200

7/33

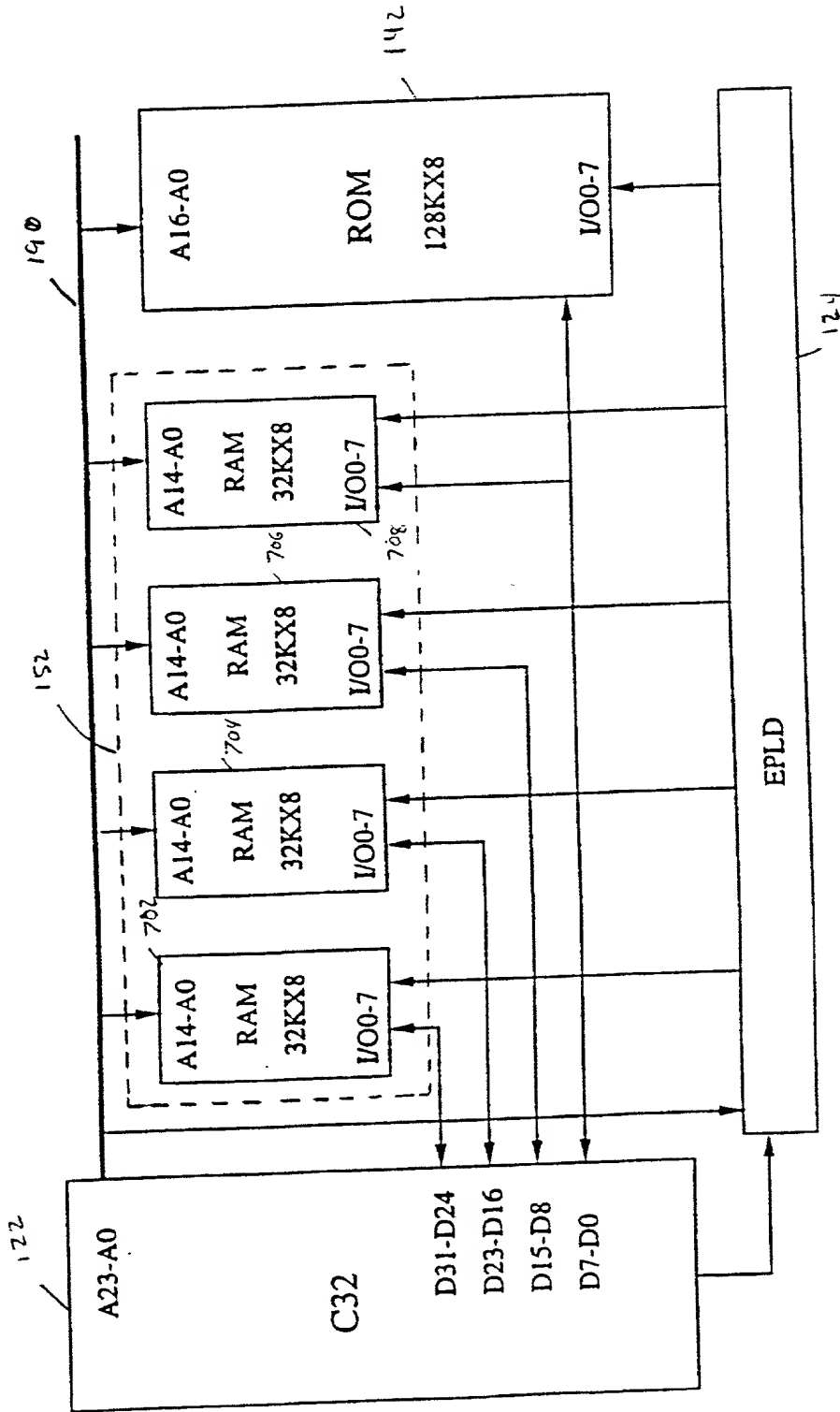


FIG. 7

20220330 53303660

8/33

800

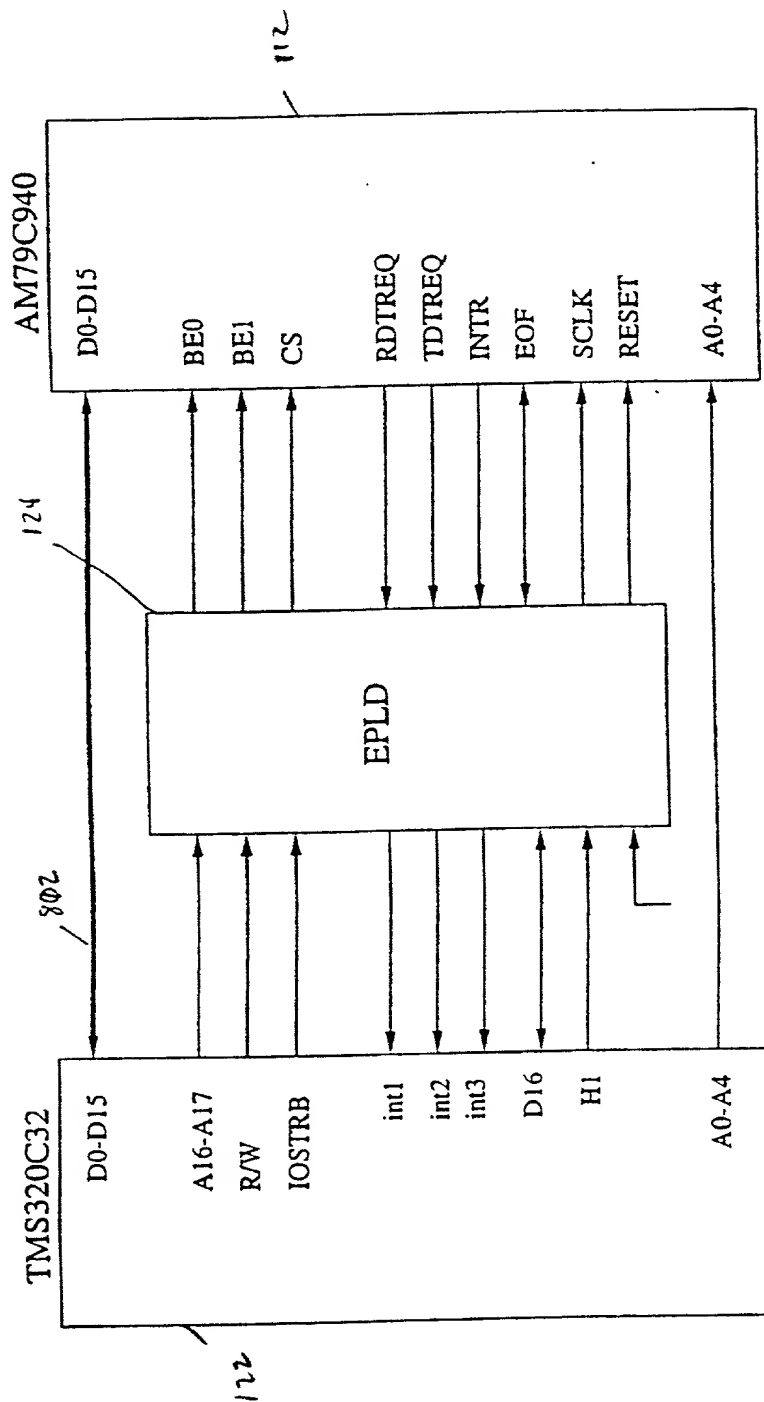


FIG. 8

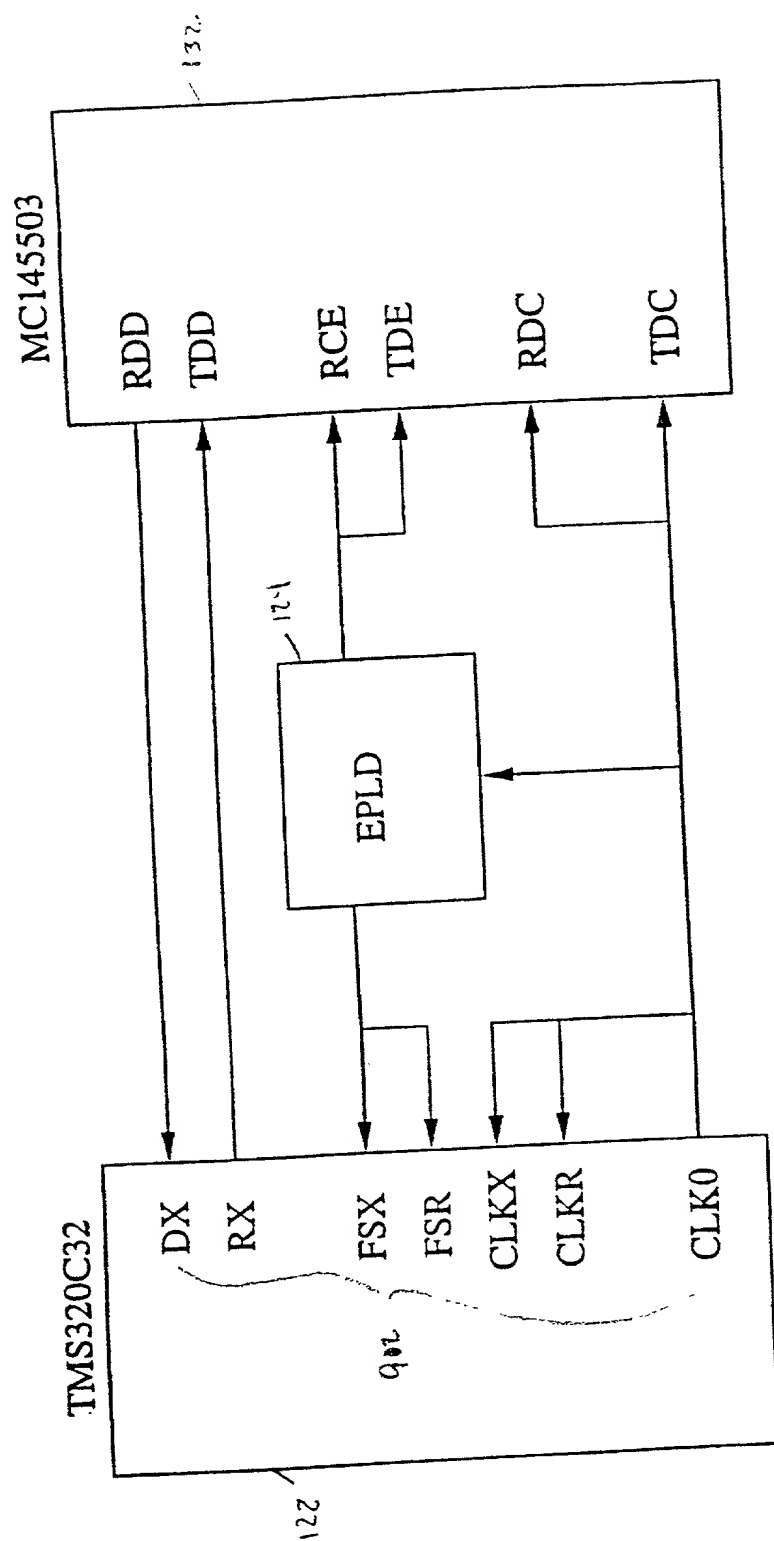
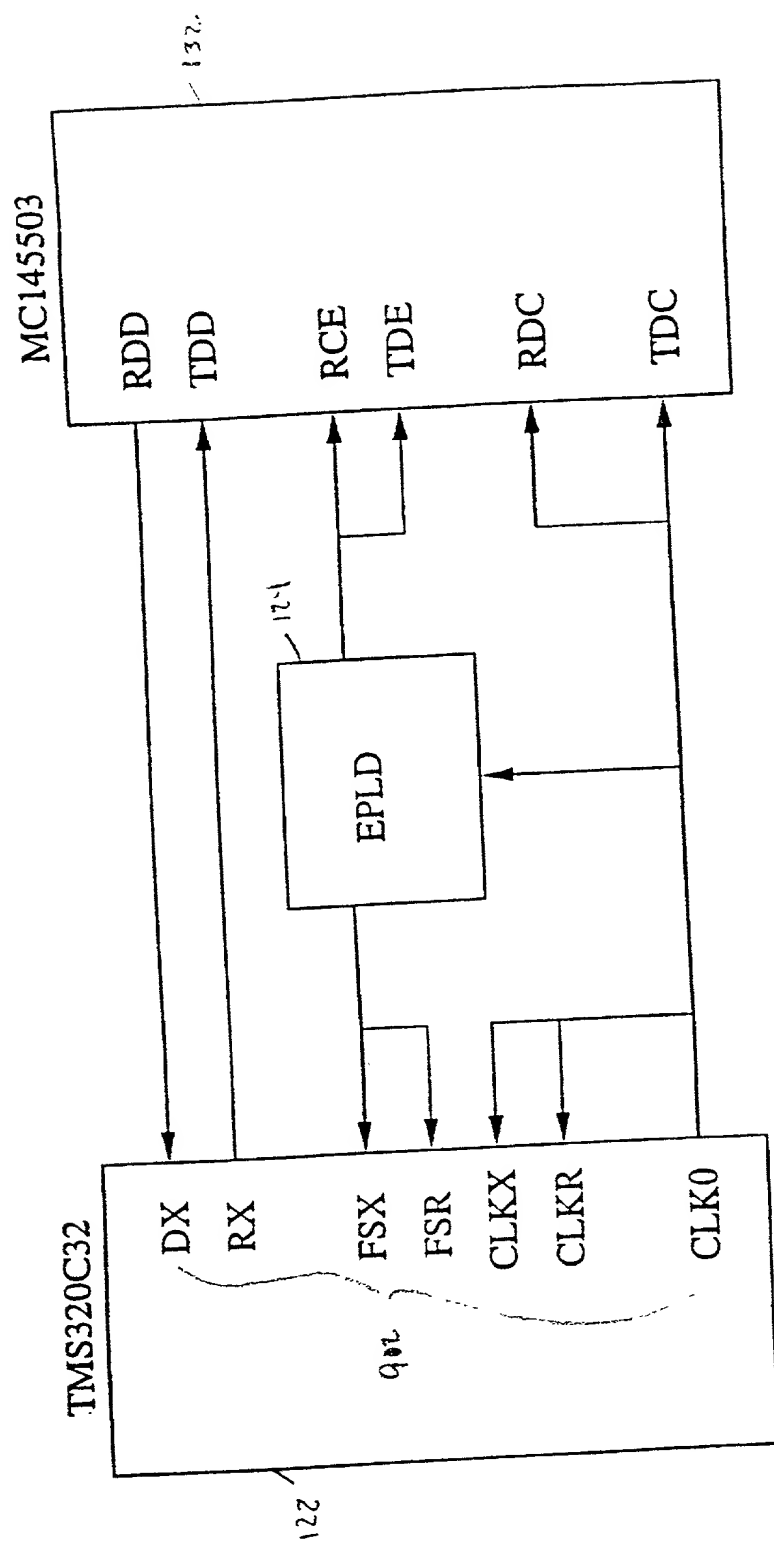


Fig. 9

10/33

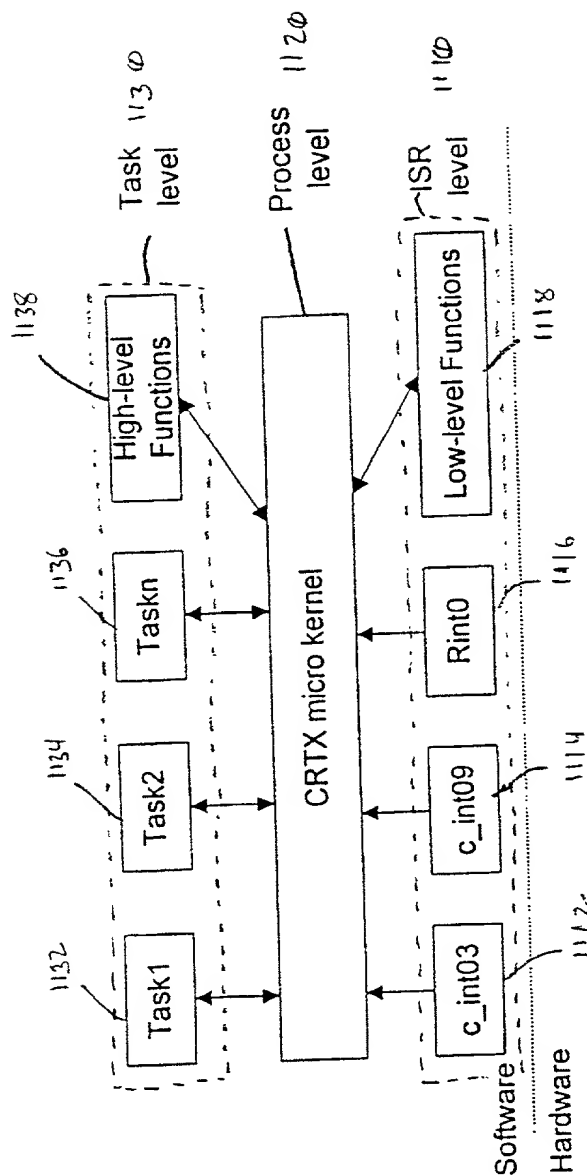
DSP Address	Usage
0x2000	Command port for left-half of LCD
0x20001	Data port for left-half of LCD
0x2002	Command port for right-half of LCD
0x2003	Data port for right-half of LCD

FIG. 10

20221211 09:03:00

11/33

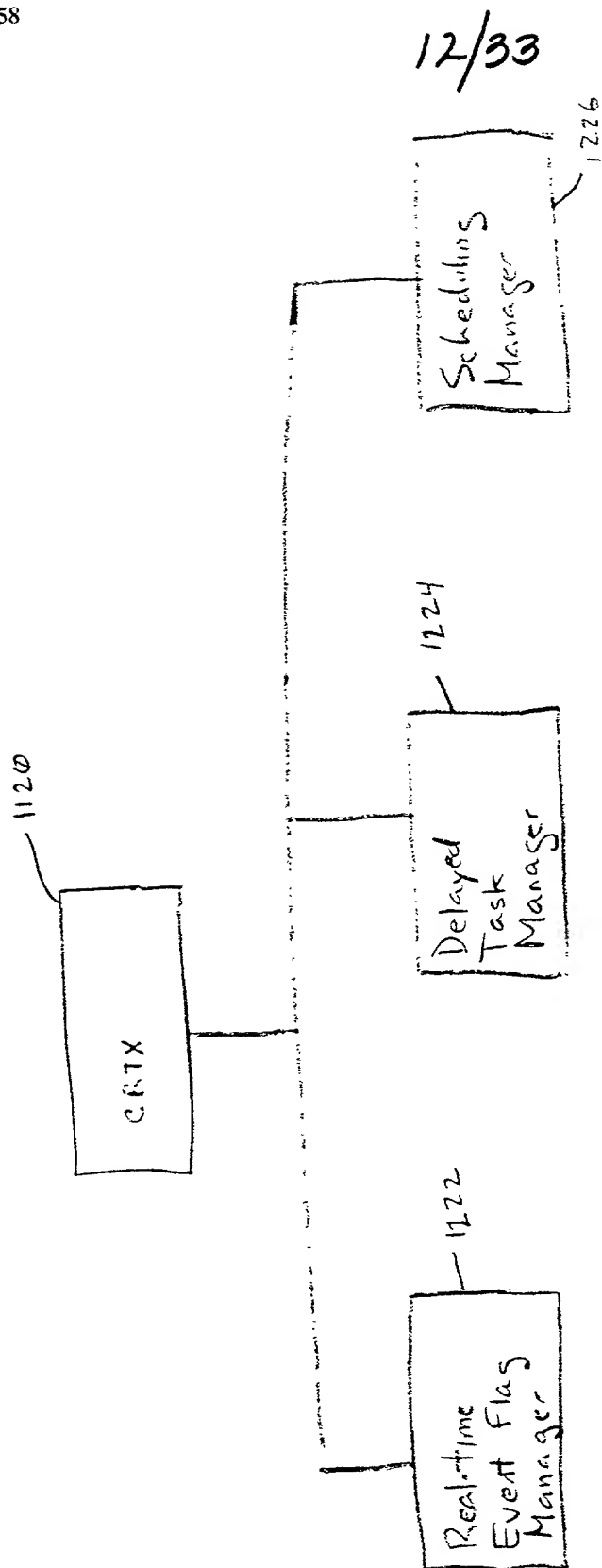
1100



{ 1100 }

[illegible]

1204



21.912

13/33

FIG. 13A

Task Level Software

Function Name	Function
<i>ARPinit()</i>	(Initialization Function) ARP table initialization.
<i>c_int00()</i>	(Initialization Function) Main program, initialize the stack pointer, external bus interface, and interrupt vector for TMS320C32.
<i>DMA_initialize()</i>	(Initialization Function) Initialize the DMA0 and DMA1 channels.
<i>ENET_initialize()</i>	(Initialization Function) Initialize the Ethernet controller.
<i>InitHardWare()</i>	(Initialization Function) Initialize the Timer0, Timer1 and serial port.
<i>NameInit()</i>	(Initialization Function) Initialize some SIP headers and SDP body.
<i>SerialPortInit()</i>	(Initialization Function) Initialize the serial port.
<i>ARP_In_task()</i>	Parse ARP input packets
<i>ARPTimer_task()</i>	ARP timer, maintain the ARP table
<i>Call_task()</i>	Call processing
<i>Clock_task()</i>	A clock generates the hour, minute, and second
<i>Codec_task()</i>	A task to call encoding, decoding, ring generation, tone generation or memory loop.
<i>CreateSipCall()</i>	Create a SIP request packet for a call
<i>Ercv_task()</i>	Ethernet packet receiver and IP de-multiplexing.

14/33

Fig. 13B

Function Name	Function
<i>IP_Send_task()</i>	IP multiplexing and Ethernet packet sending.
<i>Key_task()</i>	Key pad monitor and input.
<i>RTP_In_task()</i>	RTP processing.
<i>Sendto()</i>	Send UDP packets to given IP address.
<i>Setting_task()</i>	Setting the E*Phone parameters.
<i>SIP_In_task()</i>	Accept SIP packets, and update call and SIP status.
<i>SIP_task()</i>	SIP status transition task
<i>Tone_task()</i>	Count the active and stop duration for tone or ring.
<i>UDP_In_task()</i>	Accept UDP packets
<i>ARP_Out()</i>	(High-level function) ARP request program
<i>ClearScreen()</i>	(High-level function) Clear all lines on the LCD
<i>CodecConfig()</i>	(High-level function) Schedule a codec task according to the run mode parameter
<i>Disp()</i>	(High-level function) Display a string on the LCD screen
<i>LCD()</i>	(High-level function) Display a character on the LCD screen
<i>LCDClear()</i>	(High-level function) Clear one line on the LCD screen
<i>LinearToUlaw()</i>	(High-level function) Linear data to u-law data conversion

15/33

FIG. 13C

Function Name	Function
<i>Initialization()</i>	(High-level function) Call initialization function and pre-schedule tasks
<i>RTP_para_init()</i>	(High-level function) Generate the random time stamp and SSRC for a RTP session
<i>ScreenScroll()</i>	(High-level function) Scroll the LCD screen for one line upward or downward
<i>SDPParse()</i>	(High-level function) Parse SDP packets
<i>SIPParse()</i>	(High-level function) Parse SIP packets
<i>SIP_Request()</i>	(High-level function) Create SIP request messages
<i>SIP_Response()</i>	(High-level function) Create SIP response messages
<i>SpeechDecode()</i>	(High-level function) Speech decoding
<i>SpeechEncode()</i>	(High-level function) Speech encoding
<i>ToneGenerate()</i>	(High-level function) Generates dial tone, ring back tone, busy tone or alert tone.

17/33

FIG. 13E

ISR Level Software

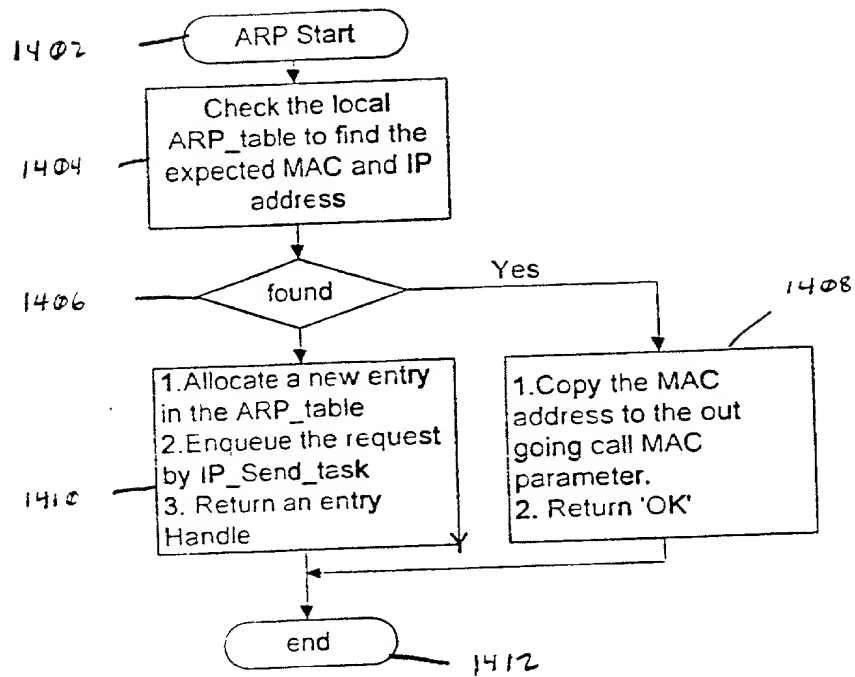
Function Name	Function
<i>c_int03()</i>	Ethernet controller ISR. Triggered on INT3 of TMS320C32 by external interrupt from AM79C940.
<i>c_int09()</i>	System timer ISR. Triggered on TINT1 by internal timer1 of TMS320C32.
<i>Rint0()</i>	A/D and D/A ISR. Triggered on RINT0 by internal serial port interrupt of TMS320C32.
<i>AmpControl()</i>	(Low-level function) Control the speaker volume.
<i>DMA1()</i>	(Low-level function) Start the DMA1 channel.
<i>DMA0_Release()</i>	(Low-level function) Start the DMA0 channel.
<i>DMA_int_set()</i>	(Low-level function) Enable INT1 and INT2 for DMA0 and DMA1.
<i>ENET_reset()</i>	(Low-level function) Reset the Ethernet controller.
<i>ENET_disable()</i>	(Low-level function) Disable the Ethernet controller.
<i>HandSet()</i>	(Low-level function) Control the handset and hands-free switching.
<i>HookState()</i>	(Low-level function) Check the hook state.
<i>Key()</i>	(Low-level function) Key pad check and read.
<i>KeyMap()</i>	(Low-level function) Map the key binary input to ASCII format.
<i>LCDCmd()</i>	(Low-level function) LCD control command.

18/33

FIG 13 F

Function Name	Function
<i>LCDWrite()</i>	(Low-level function) Write display data to LCD.
<i>RintEnable()</i>	(Low-level function) Enable the RINT0 for Rint0 ISR.
<i>RintDisable()</i>	(Low-level function) Disable the RINT0.
<i>SerialPortRst()</i>	(Low-level function) Reset the serial port.
<i>TimerEnable()</i>	(Low-level function) Enable the system timer TCLK1.
<i>TimerDisable()</i>	(Low-level function) Disable the system timer TCLK1.

19/33

1400FIG. 14

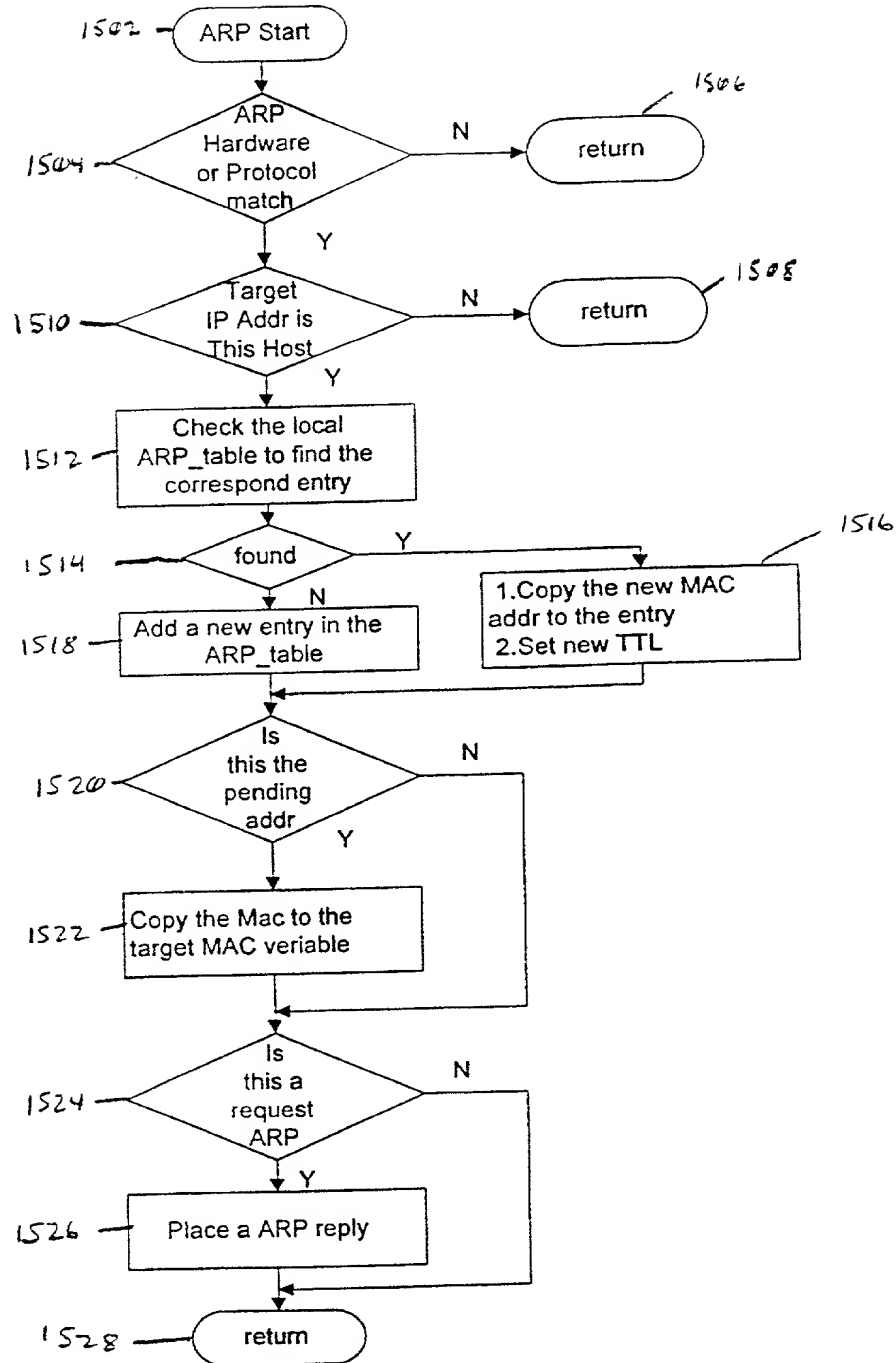
20/33
1500

FIG. 15



22/33

```

struct ENetHeader {          /* Ethernet header structure */
    ETA Dest;                 /* Ethernet destination MAC address */
    ETA Source;               /* Source MAC address */
    int Type;                 /* Ethernet packet type */
};

struct IPHeader {            /* IP header structure */
    int VI_ToS;               /* IP version, header length, and service type */
    int Length;               /* total length */
    int Identify;             /* identifier of the IP packet */
    int FragDff;              /* flags and fragment offset */
    int TTL_Protocol;         /* time-to-live, and protocols */
    int ChkSum;               /* checksum */
    IPA Source;               /* source IP address */
    IPA Dest;                 /* destination IP address */
};

struct UDPHeader {           /* UDP header structure */
    int SPort;                /* source port */
    int DPort;                /* destination port */
    int Length;               /* UDP message length */
    int ChkSum;               /* UDP checksum */
};

struct EPACKET {             /* Ethernet receive packet structure */
    struct ENetHeader Enh;     /* Ethernet header */
    struct IPHeader Iph;       /* IP header */
    struct UDPHeader Uh;       /* UDP header */
    int data[MaxUDPLength];    /* data field */
};

```

FIG. 17

202222D-58808660

23/33

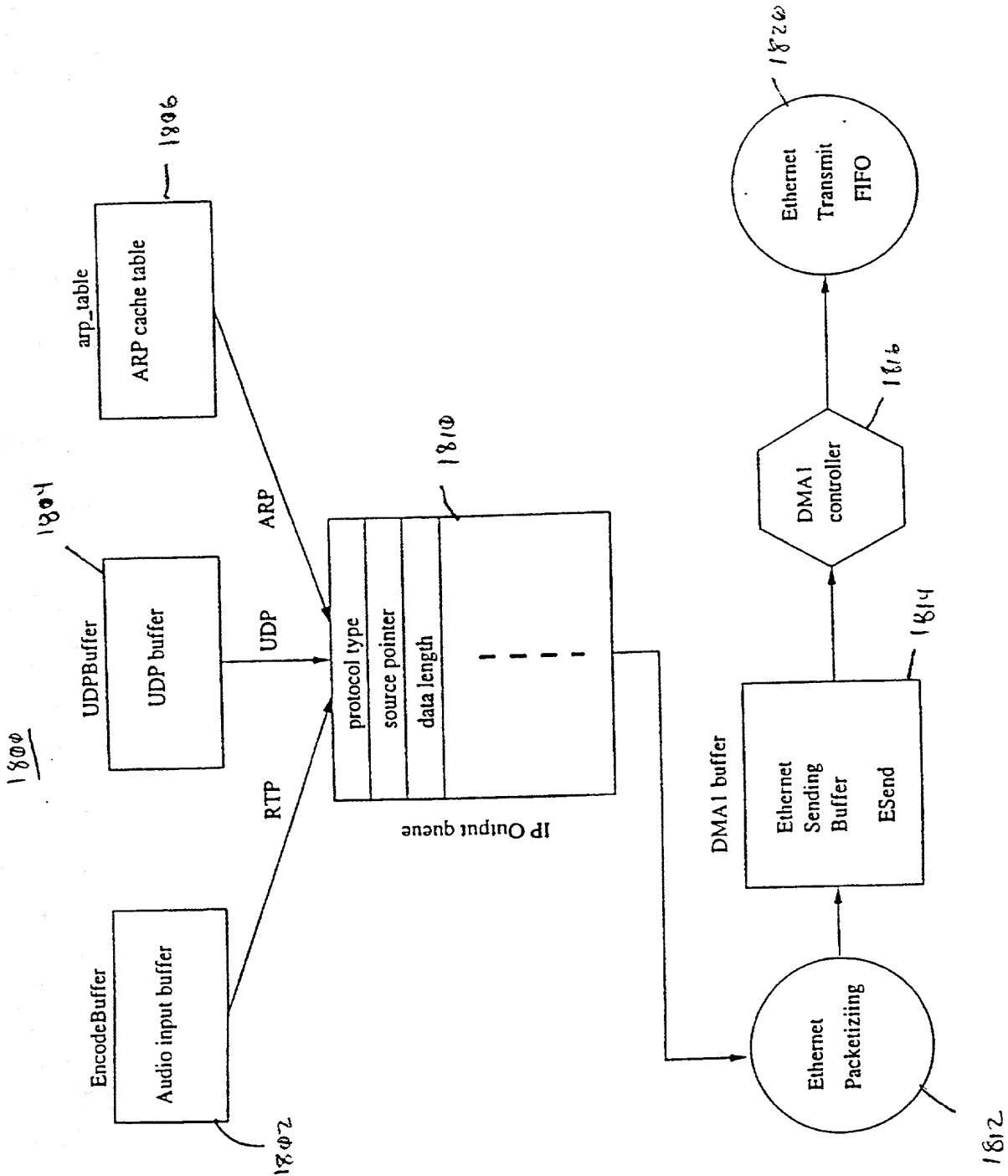
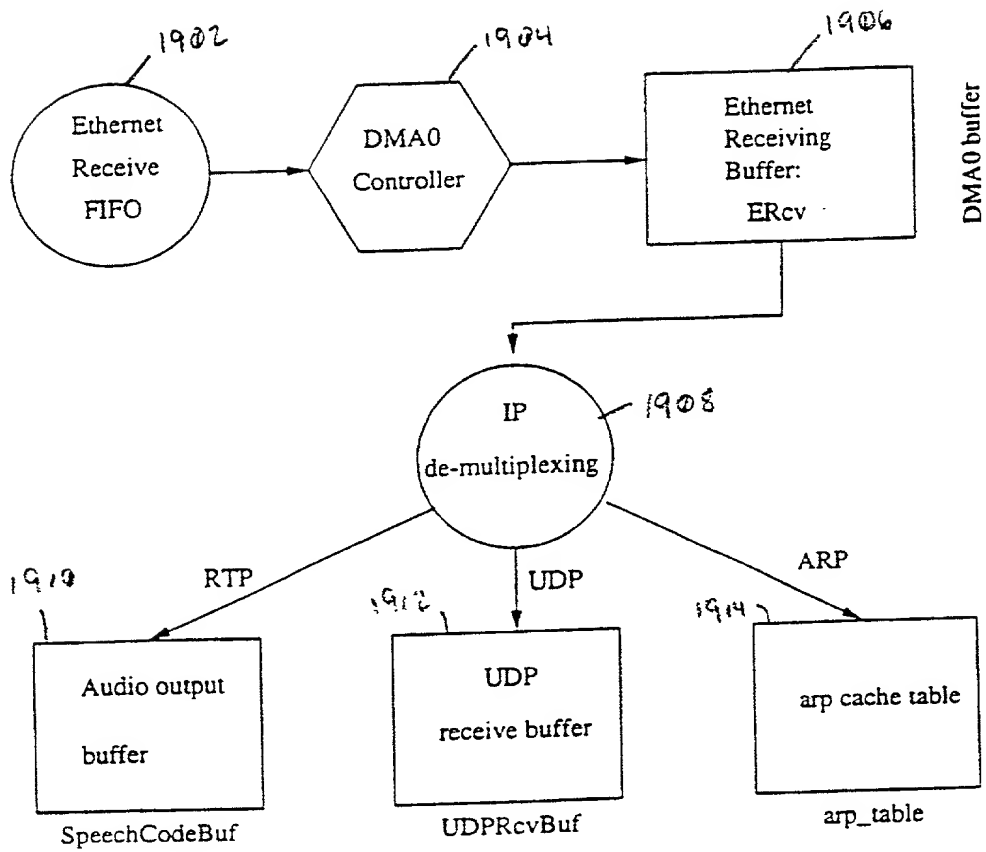


FIG. 18

24/33

1900FIG. 19

25/33

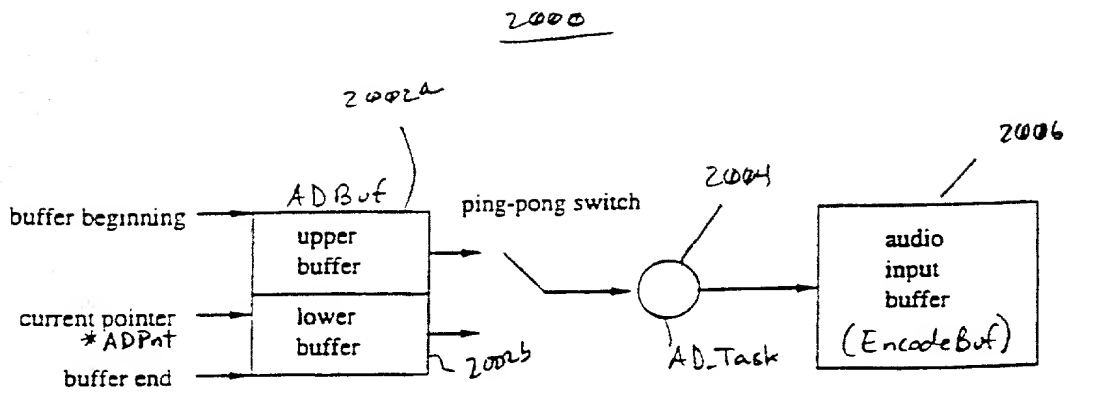


FIG. 20A

(a) A/D buffer update scheme

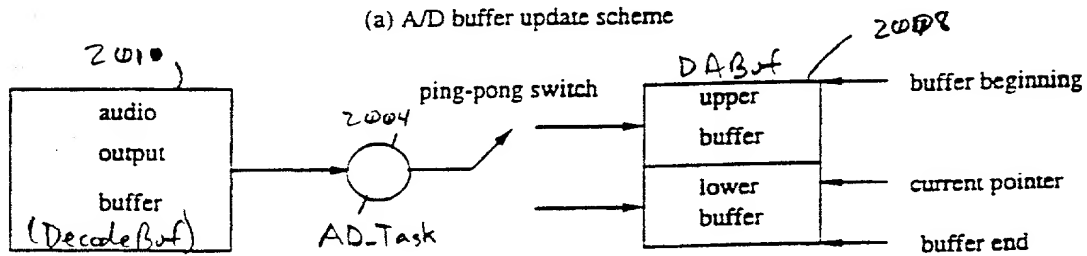
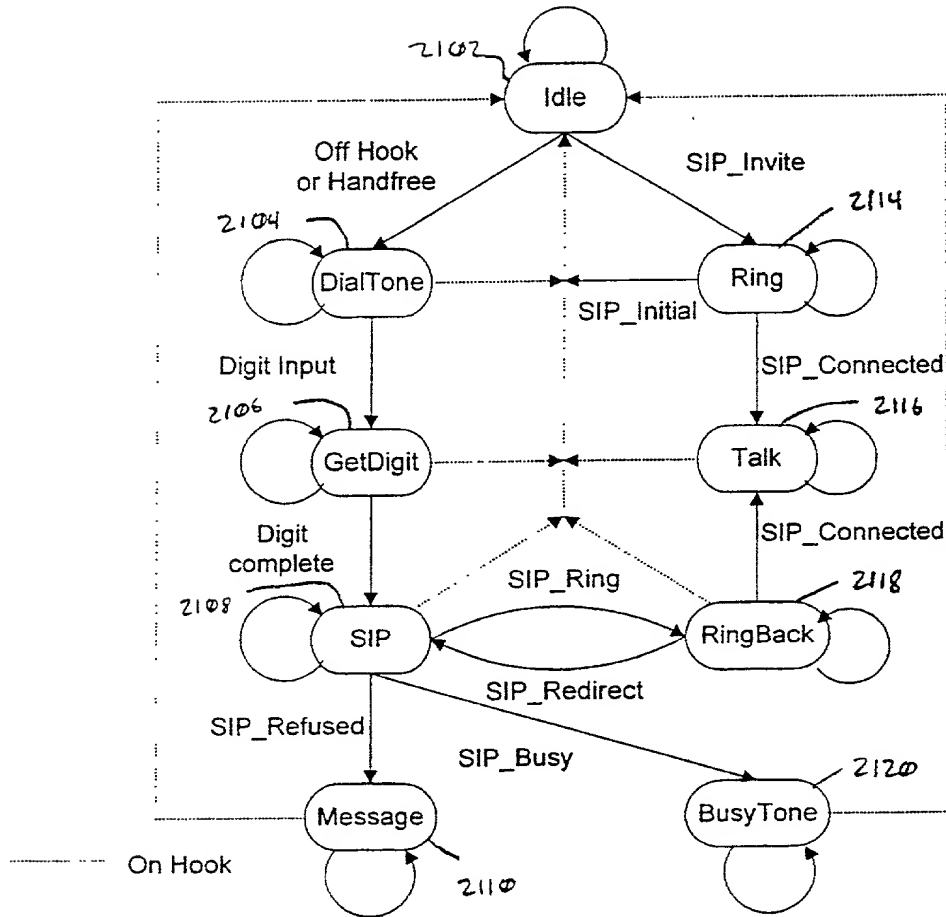


FIG. 20B

26/33

2100FIG. 21

27/33

FIG. 22

Keys	Return Value
Digit keys	'0' ... '9'
Special keys	'*' and '#'
Enter key	'E'
Hands Free	'H'
Redial key	'R'
Upward	'U'
Downward	'D'

FIG. 23

```

struct FuncKey {
    WORD Enter;      Enter key
    WORD Redial;     Redial key
    WORD Up;         Up arrow key
    WORD Down;       Down arrow key
    WORD Digit;      digit keys or special key
    WORD Full;        key buffer full
    WORD Enable;     when set, indicates key input is enabled
    WORD Touch;      any key was pressed
    WORD Alf;        an alphabetic key was pressed.
};

```

28/33

FIG 24

Port address	Read Write	Bit Value	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0x820001	W	0	x ⁵	x	x	x	Soft reset	Volume lock	ENET release	Hand free
0x820001	W	1	x	x	x	x	x	unlock	ENET reset	Hand set
0x820001	R	0	x	x	x	x	x	No key input	x	Hook off
0x820001	R	1	x	x	x	x	x	Key touched	x	Hook on

FIG. 25

```

struct Message {
    int ENetXmtST; /* a structure for all messages in the SIP Phone */
    int ENetRcvST0; /* Ethernet transmission packet state */
    int ENetRcvST1; /* Ethernet receiving packet state */
    int ENetRcvST2; /* Ethernet receiving packet state */
    int ENetRcvST3; /* Ethernet receiving packet state */
    int RcvFlag; /* Ethernet receiving packet state */
    int ARPST; /* The receiving speech data is available when SET */
}; /* reserved */

```

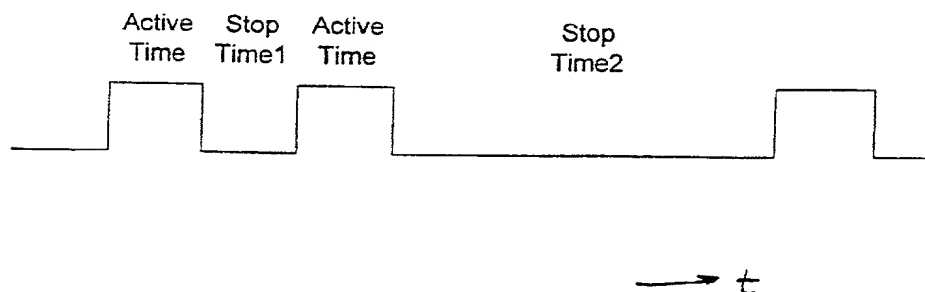
29/33
FIG. 26

```
typedef struct {
    int pt:7 /* payload type */
    int m:1 /* marker bit */
    int cc:4 /* CSRC count */
    int x:1 /* header extension flag */
    int p:1 /* padding flag */
    int version:2 /* protocol version */
    int seq /* sequence number */
    int ts1 /* timestamp least significant 16 bits */
    int ts2 /* timestamp most significant 16 bits */
    int ssrc1 /* Synchronization source least significant 16 bits */
    int ssrc2 /* Synchronization source most significant 16 bits */
    int csrc[1] /* optional CSRC list address */
} RTPHeader;
```

FIG. 27

```
struct ToneType {
    int ActiveTime; /* The period for sound is active */
    int ActiveCnt; /* The counter for the sound during the active time */
    int StopTime1; /* First sound stop period */
    int StopCnt1; /* First sound stop counter */
    int StopTime2; /* Second sound stop period */
    int StopCnt2; /* Second sound stop counter */
}
```

FIG. 28



30/33

```

typedef struct {
    char *s;
    short len;
} string;

typedef enum {
    Initial,
    Proceeding,
    Failure,
    Success,
    Confirmed,
    Calling,
    CallProc,
    Completed,
    Bye
} Tstate;

typedef struct {
    method_t method;
    short status;
    string url;
    string via;
    string callid;
    string contact;
    string from;
    string from_display;
    string subject;
    string to;
    string to_display;
    string ts;
    string reason;
    content_t contenttype;
    int contentlength;
    unsigned cseq;
    string body;
    sdp_t sdp;
} message_t;

typedef struct {
    int flag;
    int ua_state;
    int status;
    message_t m;
    char *udp;
    char *local;
    sockaddr peer;
    sdp_t sdp;
    Tstate state;
    int t1;
    int t2;
} call;

/* string type used in message_t structure */
/* start of string */
/* length of string */

/* state transition structure */
/* SIP initial state, UAC or UAS */
/* proceeding of the request, UAS */
/* failure, UAS */
/* success, UAS */
/* confirmed, UAS */
/* calling, UAC */
/* call proceeding, UAC */
/* completed, UAC */
/* Bye state, UAC or UAS */

/* SIP message structure
/* request: method; response: 0 */
/* response: status value; request: 0 */
/* request URL */
/* via header */
/* Call-ID */
/* contact header */
/* From address */
/* From display name */
/* Subject */
/* To address */
/* To display name */
/* timestamp */
/* response reason phrase */
/* contact type header */
/* contact length */
/* sequence number */
/* SDP body */
/* session description */

/* call structure */
/* SET for effective, RESET for clear */
/* Not current call:0; UAC:1; UAS:2 */
/* current response status */
/* SIP message */
/* receive SIP packets pointer */
/* UAC request packet pointer */
/* peer host IP address */
/* sdp backup */
/* SIP transition state */
/* T1 timer */
/* T2 timer */

```

FIG. 29

09980885-032202

31/33

3000

FIG. 30

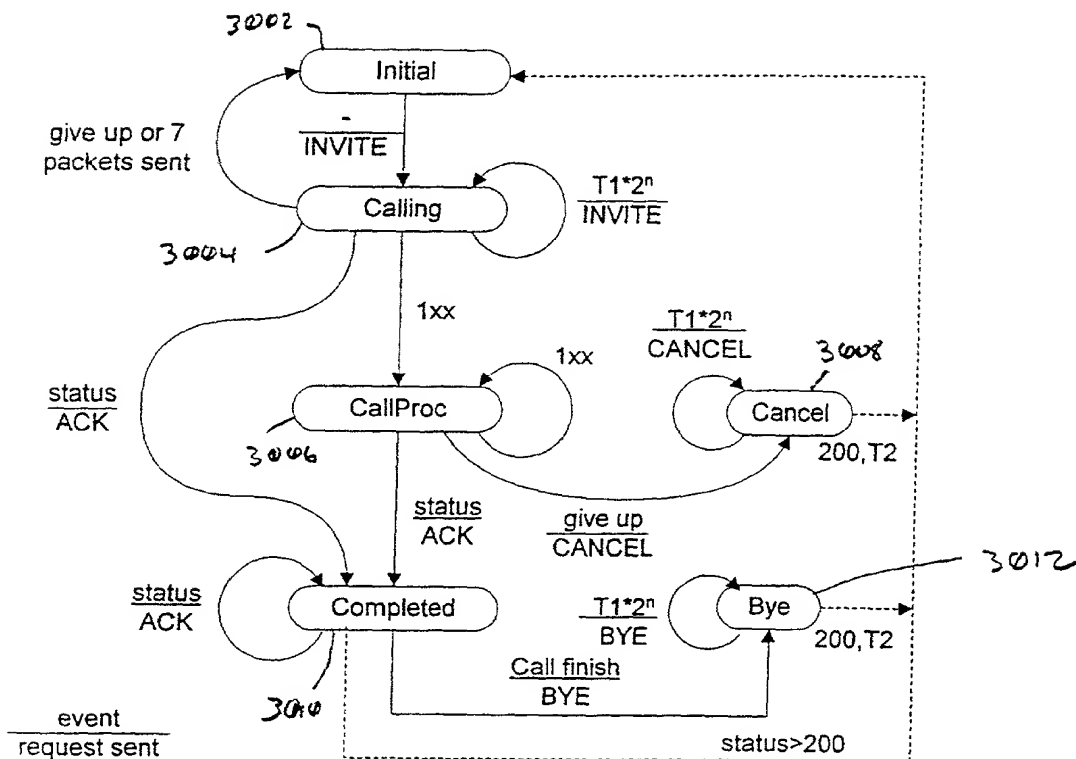
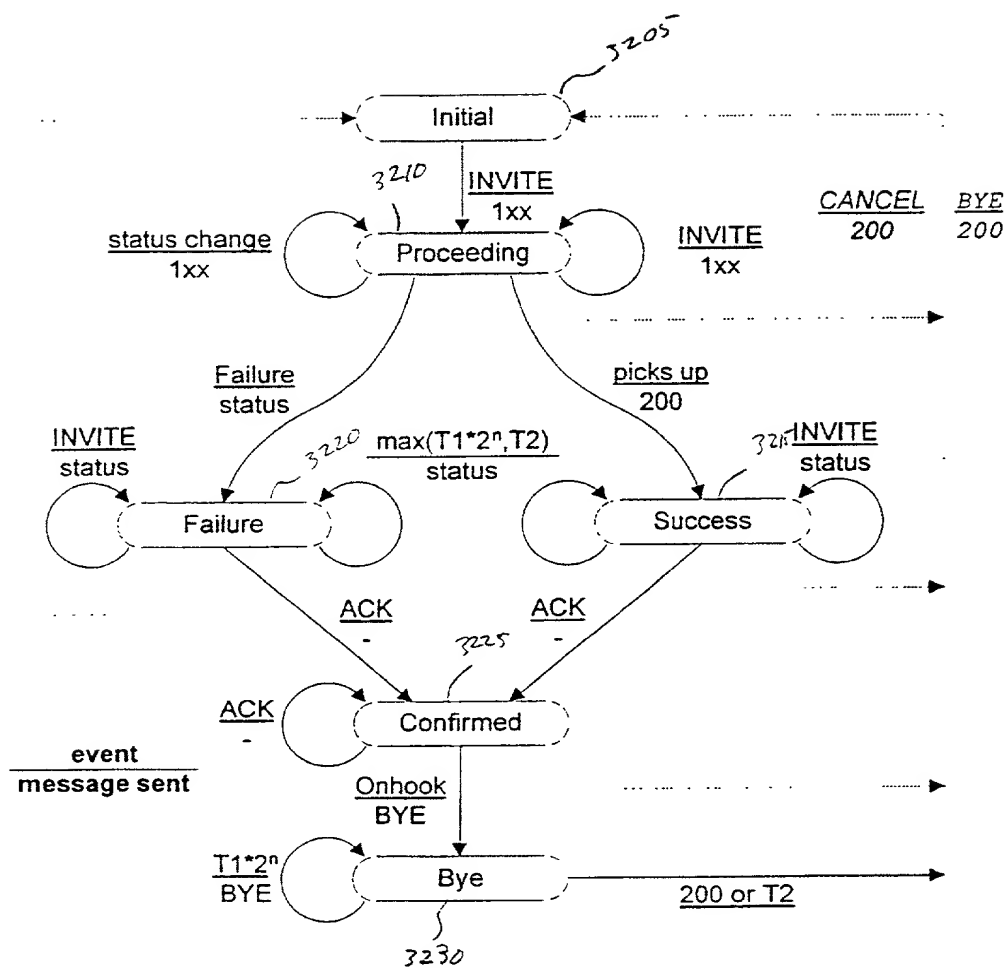


FIG. 31

Message received	SIP_Status
100	SIP_Trying
18x	SIP_Ring
200	SIP_Connected
3xx	SIP_Redirect
4xx, 5xx	SIP_Refused
6xx	SIP_Busy

32/33

FIG. 32



33/33

FIG. 33

